University of Freiburg
Institute for Computer Science
Prof. Dr. F. Kuhn

Exam Theoretical Computer Science - Bridging Course

Thursday, August 17, 2017, 10:00-12:00

Name: .....

Matriculation No.: .....

Signature: .....

Do not open or turn until told so by the supervisor!

- Write your **name** and **matriculation number** on this page and sign the document!
- Write your name on **all sheets**!
- Your **signature** confirms that you have answered all exam questions without any help, and that you have notified exam supervision of any interference.
- Write legibly and only use a pen (ink or ball point). Do **not use red**! Do **not use a pencil**!
- This is an **open book exam** therefore printed or hand-written material is allowed.
- However, **no electronic devices** are allowed.
- There are **eight tasks** (with several sub-tasks each) and there is a **total of 120 points**.
- **50 points are sufficient** in order to pass the exam.
- Only **one solution per task** is considered! Make sure to strike out alternative solutions, otherwise the one yielding the minimal number of points is considered.
- **Detailed steps** might help you to get more points in case your final result is incorrect.
- The keywords **Show...** or **Prove...** indicate that you need to prove or explain your answer carefully.
- The keywords **Give...** or **State...** indicate that you only need to provide a plain answer.
- You may use information given in a **Hint** without explaining them.
- **Read each task thoroughly** and make sure you understand what is expected from you.
- **Raise your hand** if you have a question regarding the formulation of a task.
- **Use the space below each task and the back of the sheet for your solution.** The last two sheets of this exam are blank and can be used for solutions. If you need additional sheets, raise your hand.

Question	1	2	3	4	5	6	7	8	**Total**
Points									
Maximum	14	17	15	14	15	13	16	16	120

Task 1: Basic Mathematical Skills (14 Points)

(a) Prove the equation $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ for all $n \in \mathbb{N}_0$ **by induction** on n. *(5 Points)*

(b) Let A, B be sets. We define the symmetric difference $A \,\Delta\, B := (A \setminus B) \cup (B \setminus A)$.
Prove the following **implication**: *(4 Points)*

$$A \cap B \neq \emptyset \quad \Longrightarrow \quad A \,\Delta\, B \neq A \cup B.$$

***Remark**: $A \setminus B := A \cap \overline{B}$ is the 'set minus' operator, describing all elements of A that are not in B. Instead of a formal proof, you can show the implication with Venn diagrams.*

(c) Give the **minimum and the maximum number of edges** an undirected, **bipartite** graph $G = (V, E)$ with $n := |V|$ nodes can have. You may assume that n is even. *(1+4 Points)*

***Remark**: A graph $G = (V, E)$ is **bipartite** if its nodes V can be partitioned into two disjoint sets $U, W \subseteq V$, such that there are **no** edges in E among any two nodes in U, and the same is true for W. That is, for all $\{v_1, v_2\} \in E$ it holds that v_1 and v_2 are not in the same part.*

Sample Solution

(a) *Induction base:* The statement is true for $n = 0$, since $\sum_{i=0}^{0} 2^i = 1 = 2^{0+1} - 1$. *(1 Points)*

Induction hypothesis: Presume the statement holds for an arbitrary, fixed $n \in \mathbb{N}_0$.

Induction step: $\sum_{i=0}^{n+1} 2^i = 2^{n+1} + \sum_{i=0}^{n} 2^i \overset{\text{Ind. hyp.}}{=} 2^{n+1} + 2^{n+1} - 1 = 2^{n+2} - 1$. *(4 Points)*

(b) Since $A \cap B \neq \emptyset$ there is an $x \in A \cap B \subseteq A \cup B$. Since $x \in A$ and $x \in B$, we have $x \notin A \setminus B$ and $x \notin B \setminus A$. Therefore $x \notin (A \setminus B) \cup (B \setminus A) = A \,\Delta\, B$.

(c) *Minimum number:* 0. *(1 Points)*

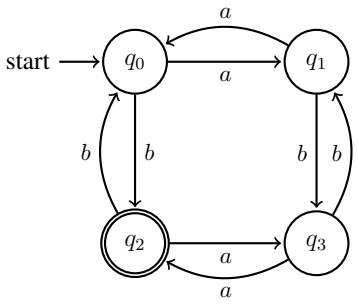
Reasoning: (not required for full points) A graph without any edges is obviously bipartite.

Maximum number: $(\frac{n}{2})^2$ if n is even (and $\lceil n/2 \rceil \cdot \lfloor n/2 \rfloor$ if n is odd). *(4 Points)*

Reasoning: (not required for full points) a bipartition U, W with $m := |U|$ and $k := |W|$ can have at most $m \cdot k$ edges. The term $m \cdot k$ can be interpreted as the area of a rectangle (subject to $m + k = n$) which is maximized for equal, or almost equal, side length.

Task 2: Regular Languages (17 Points)

Consider the following *Deterministic Finite Automaton (DFA)* A over the alphabet $\{a, b\}$.



(a) Give the **shortest string** accepted by A. *(1 Points)*

(b) Give an **infinite** set of strings that are accepted by A and **consist only of *b*'s**. *(3 Points)*

(c) **Describe the language** $L(A)$ recognized by A (as a set or verbally). *(4 Points)*

(d) Now consider the language $L(\gamma)$ given by the regular expression $\gamma := (ab)^*(ba)^*$.

Give a DFA that recognizes $L(\gamma)$ and has **at most four states**. *(6 Points)*

***Remark**: You can give a non-deterministic finite automaton (NFA) for a penalty of 2 points or an automaton with more than four states for a penalty of 1 point for each additional state.*

(e) Let L be the language consisting of words of the form $w_1w_2w_3$ with $w_1, w_2, w_3 \in \{a, b, c\}^*$ **and** w_1 contains **no** a's **and** w_2 contains **no** b's **and** w_3 contains **no** c's.

Give a regular expression that generates L. *(3 Points)*

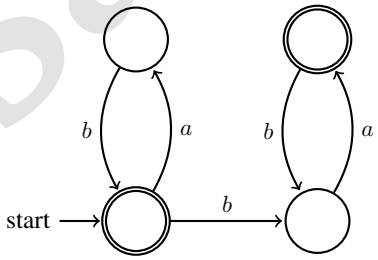
Sample Solution

(a) The shortest string in $L(A)$ is b.

(b) A accepts the family $\{b(bb)^n \mid n \in \mathbb{N}_0\}$.

(c) $L(A) := \{w \in \{a, b\}^* \mid$ the number of a's in w is even and the number of b's in w is odd$\}$.

(d)



(e) $(b \cup c)^* \cdot (a \cup c)^* \cdot (a \cup b)^*$.

Task 3: Context-Free Languages (15 Points)

Let $L = \{ss^{RC} \mid s \in \{0,1\}^*\}$ be a language over alphabet $\{0,1\}$, where s^{RC} describes the **reverse complement** of a string $s \in \{0,1\}^*$, obtained by reversing the order of symbols in s and then exchanging every 0 in s with 1 and every 1 in s with 0.

(a) State whether there is a string in L with an **unequal number of zeros and ones**. *(1 Points)*

(b) Give a **context-free grammar** that generates L. *(3 Points)*

(c) Give a **Pushdown Automaton** (PDA) that recognizes L. *(5 Points)*

(d) Prove that L is **not a regular language** by using the **Pumping Lemma**. *(6 Points)*

Sample Solution

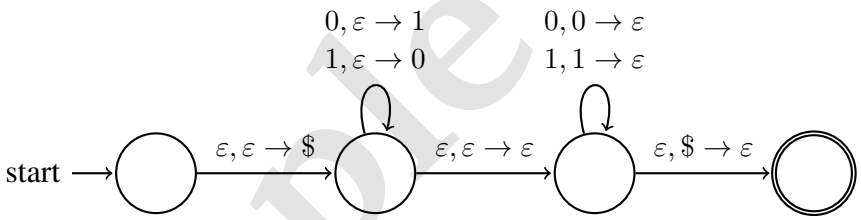
(a) No, there exists no such string in L since $|s|_0 = |s^{RC}|_1$ and $|s|_1 = |s^{RC}|_0$, hence

$$|ss^{RC}|_0 = |s|_0 + |s^{RC}|_0 = |s^{RC}|_1 + |s|_1 = |ss^{RC}|_1.$$

Remark: Just 'No' suffices.

(b) Let $G = (\{S\}, \{0,1\}, R, S)$ with $R := \{S \to \varepsilon | 0S1 | 1S0\}$. Then $L(G) = L$.

(c)



(d) We prove that L is not regular by showing that L violates the pumping condition, i.e. always contains a string that can not be pumped. Let p be the pumping length. We investigate the string $s = 0^p1^p$. Since $0^p = (1^p)^{RC}$ we have $s \in L$. Furthermore s is longer than p.

Consider a partition $xyz = s$ with $|y| \geq 1$ and $|xy| \leq p$. It follows that $y = 0^q$ with $1 \leq q \leq p$. Since $xy^0z = 0^{p-q}1^p$ has fewer zeros than ones, we can not divide xy^0z into a string and its reverse complement, because any string of this form must have a number of zeros equal to the number of ones. Hence $xy^0z \notin L$. *Remark: Works with any choice $i \neq 1$.*

Task 4: Turing machines (14 Points)

(a) Give a **comparison** of the set of languages recognized by **deterministic** Turing machines with the set of languages recognized by **non-deterministic** Turing machines. *(2 Points)*

(b) State **two differences** between **deterministic** and **non-deterministic** Turing machines.

***Remark**: You obtain 1 point for the first difference and 2 points for the second. (1+2 Points)*

(c) One can define a variant of the Turing machine which allows **three** actions of the read/write-head: $\boldsymbol{\{L, R, S\}}$, where $\boldsymbol{S}$ means that the head stands still during that step.

Let M_1 be a Turing machine **that uses head movements $\boldsymbol{\{L, R, S\}}$**. Give an **explicit** construction procedure that transfers M_1 into a Turing machine M_2 **that uses only head movements $\boldsymbol{\{L, R\}}$** and recognizes the same language, i.e. $L(M_1) = L(M_2)$. *(5 Points)*

(d) **Briefly explain** how to construct (or explicitly construct) a Turing machine for the language defined by the automaton depicted in Task 2 of this exam. *(4 Points)*

Sample Solution

(a) As proven in the lecture, deterministic and non deterministic Turing machines are equally powerful in terms of the languages they recognize. Hence **both language classes are equal**. *Remark: The bold sentence suffices to receive full points.*

(b) Non-exhaustive list of possible namings of differences:

1. Non-deterministic Turing machines might define multiple transitions for the same state and symbol read by the read/write-head.
2. The transition *function* δ of deterministic Turing machines is a transition *relation* for non-deterministic Turing machines.
3. Non-deterministic Turing machines may have ε-transitions.
4. Non-deterministic Turing machines *probably* recognize more languages in polynomial time than deterministic ones (if $\mathcal{P} \neq \mathcal{NP}$).
5. Acceptance is defined differently. In the case of non-deterministic Turing machines the existence of an accepting path suffices to recognize an input, even though other paths for that input may not be accepting.

(c) We take the TM machine M_1 and subsequently remove all transitions which contain the neutral movement until we obtain a Turing machine according to the standard definition. For each $\delta(q, a) = (q', b, S)$ add an additional state $\tilde{q} = \tilde{q}(q, a, q', b, S)$. Then remove this transition and add transitions $\delta(q, a) = (\tilde{q}, b, L)$ and $\delta(\tilde{q}, x) = (q', x, R)$ for all $x \in \Sigma$ instead.

Reasoning: (not required for full points) After execution of these two transitions the Turing machine will be in the same configuration as it would have been when executing the original transition. Each step of M_1 can be simulated with at most two steps by M_2.

(d) We modify the DFA from Task 2 such that it becomes a Turing machine. The automaton stays the same we change only the transitions to the format of a Turing machine. This can be done by moving the read/write-head to right over each input symbol for each transition of the DFA. I.e. a transition $\delta(q, a) = q'$ of the DFA becomes a transition $\delta'(q, a) = (q', a, R)$.

Remark: A short informal explanation how to transform A from Task 2 into an equivalent Turing machine suffices. Formally, for the DFA $A = (Q, \Sigma, \delta, q_0, \{q_2\})$ we define the Turing machine $M = (Q, \Sigma, \Gamma = \Sigma \cup \{\sqcup\}, \delta', q_0, q_{accept} = q_2, q_{reject}\})$ where q_{reject} is the implicit garbage state of A. This procedure can be applied to any DFA, provided it has only one final state (however we can just as easily define Turing machines with multiple final states). Giving an explicit state diagram of a Turing machine M with $L(M) = L(A)$ is also accepted.

Task 5: $\mathcal{O}$ - Notation (15 Points)

State whether the following claims are true or false *(1 point each)*. Then **prove or disprove** the claim *(6 points for (a) and 7 points for (b))*. Use the definition of the $\mathcal{O}$-notation.

(a) $n^{\sqrt{2}} \in \mathcal{O}(\sqrt{2} \cdot n)$. ***Hint:*** $\sqrt{2} > 1$. *(1+6 Points)*

(b) $2^{\sqrt{n}} \in \mathcal{O}((\sqrt{2})^n)$. *(1+7 Points)*

Sample Solution

(a) The claim is false. We disprove it by showing a contradiction. Assume there is a constant $c > 0$ and $M \in \mathbb{N}$ such that for all $n > M$ the inequality $n^{\sqrt{2}} \leq c \cdot \sqrt{2} \cdot n$ holds.

$$
\begin{aligned}
& n^{\sqrt{2}} \leq c \cdot \sqrt{2} \cdot n \\
\iff \quad & \frac{n^{\sqrt{2}}}{n} \leq c \cdot \sqrt{2} \\
\iff \quad & n^{\overbrace{\sqrt{2}-1}^{>0 \text{ (Hint)}}} \leq c \cdot \sqrt{2}
\end{aligned}
$$

However the left side $n^{\sqrt{2}-1}$ is unbounded, while the right side is constant. Therefore the inequality is false for all sufficiently large n, a contradiction.

(b) The claim is true. We give $c > 0$ and $M \in \mathbb{N}$ such that for all $n \geq M$: $2^{\sqrt{n}} \leq c \cdot (\sqrt{2})^n$.

$$
\begin{aligned}
& 2^{\sqrt{n}} \leq c \cdot (\sqrt{2})^n \\
\iff \quad & \log(2^{\sqrt{n}}) \leq \log(c \cdot (\sqrt{2})^n) \\
\iff \quad & \sqrt{n} \cdot \log 2 \leq \log c + n \cdot \log \sqrt{2} \\
\iff \quad & \sqrt{n} \leq \log c + \frac{n}{2} \\
\overset{c=1}{\iff} \quad & 1 \leq \frac{\sqrt{n}}{2}
\end{aligned}
$$

Which is true for $c = 1$ and all $n \geq M := 4$.

Task 6: Decidability (13 Points)

(a) Consider the problem COLORING:

$$\text{COLORING} := \{\langle G, k\rangle \mid \text{undirected graph } G \text{ has a } \textbf{k-coloring}\}.$$

A **k-coloring** of $G = (V, E)$ is an assignment $c : V \to \{1, \ldots, k\}$ of nodes to colors, such that no equally colored nodes are adjacent, i.e., for all edges $\{u, v\} \in E$ we have $c(u) \neq c(v)$.

(i) Show that COLORING is **decidable** by giving an algorithm (abstract description or pseudo-code) that decides whether a graph has a k-coloring. *(6 Points)*

(ii) Explain why your algorithm accepts **exactly** the instances $\langle G, k\rangle$ which have a k-coloring and why it always halts. *(2+1 Points)*

(b) Consider the problem MULTIPARTITION

$$\text{MULTIPARTITION} := \{\langle G, k\rangle \mid \text{undirected graph } G \text{ has a } \textbf{k-partition}\}.$$

A **k-partition** of $G = (V, E)$ is a partition of V into k disjoint subsets $V_1, \ldots, V_k$ such that there are no edges among nodes from two different subsets. Formally: For all edges $\{u, v\} \in E$ it holds that u and v are in different subsets, i.e., $u \in V_i, v \in V_j$ with $i \neq j$.

A decider for COLORING can be used to show the decidability of MULTIPARTITION.

Explain how to use your algorithm for COLORING to decide MULTIPARTITION. *(4 Points)*

***Remark**: If you did not succeed in giving an algorithm that decides* COLORING *in (a), you may assume that you have such an algorithm.*

Sample Solution

(a) Given an input graph $G = (V, E)$ with input number k, we test for all of the at most $\mathcal{O}(k^{|V|})$ possible assignments $c : V \to \{1, \ldots, k\}$ of colors to nodes, whether they form a proper coloring. *Remark: The exact number of possible assignments does not matter, it is sufficient to state that there are only finitely many.*

To check whether an assignment c is a proper coloring, the algorithm tests for all edges $\{u, v\} \in E$ whether $c(u) \neq c(v)$. If we find an edge $\{u, v\} \in E$ with equally colored end-nodes, i.e. $c(u) = c(v)$, the algorithm drops the current color assignment and continues with the next one.

If all edges were checked positively, i.e. $c(u) \neq c(v)$ for all $\{u, v\} \in E$ the algorithm accepts and halts. If, on the other hand, all possible assignments were dropped due to equally colored, adjacent nodes, the algorithm rejects. *(6 Points)*

If instance $\langle G, k\rangle$ has a k-coloring, the algorithm will accept it at some point, since the algorithm tries *all* possible color assignments and eventually finds the valid k-coloring. Conversely, if G has no valid k-colorings all color assignments are dropped due to equally colored, adjacent nodes, and the algorithm rejects $\langle G, k\rangle$. *(1+1 Points)*

A check whether $c(u) \neq c(v)$ can be done in $\mathcal{O}(|V|)$ time. For each color assignment c we do at most $\mathcal{O}(|E|)$ of those checks. Furthermore we have at most $\mathcal{O}(k^{|V|})$ possible color assignments with at most k colors. Therefore the algorithm terminates after $\mathcal{O}(|V| \cdot |E| \cdot k^{|V|})$ steps, i.e. finitely many steps. *(1 Points)*

Remark: The run-time of the algorithm does not need to be analyzed in order to obtain the point. It suffices to point out that all involved steps are finished after finite time.

(b) A k-coloring induces a k-partition (and vice versa). This can be done by putting nodes v with the same color $c(v) \in \{1, \dots k\}$ into the respective subset $V_{c(v)}$. Since nodes of a color i are not adjacent in a proper k-coloring, there are no adjacent nodes in V_i. The process works also in the opposite direction. **Therefore G has a k-coloring if and only if G has a k-partition**. Hence, we can decide whether G has a k-partition by simply deciding whether it has a k-coloring, with our algorithm from (a).

*Remark: The vital part is stating the equivalency (**bold**), which yields at least 3 points if done properly. One more point if there is some reasonable explanation. At most 1 point for stating only one direction, e.g., the implication "k-coloring $\rightsquigarrow$ k-partition", (not sufficient for a reduction).*

Task 7: Complexity Theory (16 Points)

(a) Give a language which is **in $\mathcal{NP}$ but not in $\mathcal{P}$**. Assume that $\mathcal{P} \neq \mathcal{NP}$! *(2 Points)*

(b) Give a language which is **neither in $\mathcal{P}$ nor in $\mathcal{NP}$**. *(2 Points)*

(c) Given a set U of n elements ('universe') and a collection $S \subseteq 2^U$ of m subsets of U, a selection $C_1, \ldots, C_k \in S$ of k sets is called a **set cover** of size k if $C_1 \cup \ldots \cup C_k = U$. The SETCOVER-problem is defined as

$$\text{SETCOVER} := \{\langle U, S, k\rangle \,|\, U \text{ is a set, } S \subseteq 2^U \text{ and there is a } \textbf{set cover} \text{ for } (U, S) \text{ of size } k\}.$$

Assume that we already know that the problem VERTEXCOVER is $\mathcal{NP}$-complete

$$\text{VERTEXCOVER} := \{\langle G, k\rangle \mid \text{undirected graph } G \text{ has a } \textbf{vertex cover} \text{ of size at most } k\}.$$

Given a graph $G = (V, E)$, a **vertex cover** is a subset $V' \subseteq V$ of nodes of G such that every edge of G is adjacent to a node in the subset V'.

Show that SETCOVER is $\mathcal{NP}$-complete. *(12 Points)*

***Hint**: For the polynomial reduction, **let the edges E** of a given instance of the* VERTEXCOVER *problem **be the universe U** for the associated instance of the* SETCOVER *problem.*

Sample Solution

(a) VERTEXCOVER is in $\mathcal{NP}$ and due to its $\mathcal{NP}$-completeness not in $\mathcal{P}$ subject to $\mathcal{NP} \neq \mathcal{P}$.

(b) The halting problem is neither in $\mathcal{P}$ nor in $\mathcal{NP}$ as it is undecidable and all problems in $\mathcal{NP}$ and $\mathcal{P}$ are decidable.

(c) First we show SETCOVER $\in \mathcal{NP}$ using the *'guess and check'* procedure.

Guess: Given an instance (U, S) of SETCOVER, guess a selection $C_1, \ldots, C_k \in S$ of k sets.

Check: Given a selection $C_1, \ldots, C_k \in S$ we give an algorithm that checks if $C_1 \cup \ldots \cup C_k = U$ (or not!). **For each $u \in U$ we check whether u is contained in at least one of the C_i.**

We can do this with a loop over all elements $u \in U$. Then we use a second loop nested inside the first one, looping from $i = 1$ to k over the C_i. The check whether $u \in C_i$ can be done with a third loop nested inside the second, that loops over all $c \in C_i$ and checks if $c = u$.

If we find a $u \in U$ which is not contained in any C_i, then the checking algorithm immediately rejects the guessed solution (we need make sure that the non-deterministic 'guessing machine' can not 'sell' us a wrong guess). Otherwise we accept at the end.

The runtime for this algorithm is at most $\mathcal{O}\big(|U| \cdot k \cdot \max_{i=1,\ldots,k} |C_i|\big) \subseteq \mathcal{O}\big(k|U|^2\big)$, which is polynomial in the input size. *(4 Points)*

Remark: It is not necessary to give a detailed algorithm description or even pseudo code for the checking process to receive full points. A quick explanation (like the bold sentence) and a rough polynomial upper bound for the runtime of the checking procedure suffices.

Second we reduce VERTEXCOVER to SETCOVER to show that SETCOVER is $\mathcal{NP}$-hard.

Let $\langle G, k\rangle$ (with $G = (V, E)$) be a given VERTEXCOVER instance. We reduce it to a corresponding SETCOVER instance $\langle U_G, S_G, k\rangle$ as follows: Let $U_G := E$ (Hint!) and $S_G := \{C_v \mid v \in V\}$ where $C_v := \{e \in E \mid v \in e\} \subseteq U_G$ is the set of adjacent edges of v.

The construction of $\langle U_G, S_G, k\rangle$ can be done in polynomial time (in the input size $|\langle G, k\rangle|$).

Now we need to show that $\langle G, k\rangle \in$ VERTEXCOVER if and only if $\langle U_G, S_G, k\rangle \in$ SETCOVER.

$$
\begin{aligned}
\langle G, k\rangle \in \text{VERTEXCOVER} &\Leftrightarrow G \text{ has a } \textit{vertex cover } V' = \{v_1, \ldots, v_k\} \subseteq V \text{ of size } k \\
&\Leftrightarrow \text{Every edge } e \in E \text{ is adjacent to one of the nodes } v_1, \ldots, v_k \\
&\Leftrightarrow C_{v_1} \cup \ldots \cup C_{v_k} = E \\
&\Leftrightarrow C_{v_1}, \ldots, C_{v_k} \text{ is a } \textit{set cover} \text{ of } (U_G, S_G) \\
&\Leftrightarrow \langle U_G, S_G, k\rangle \in \text{SETCOVER}.
\end{aligned}
$$

(8 Points)

Task 8: Logic (16 Points)

(a) Consider the following propositional formula

$$\psi := (x \wedge y \to z \vee w) \wedge (y \to x) \wedge (z \wedge y \to 0) \wedge (w \wedge y \to 0) \wedge y.$$

(i) Transfer ψ into an equivalent formula in **conjunctive normal form (CNF)**. *(3 Points)*

(ii) Use the **resolution calculus** to show that ψ is unsatisfiable. *(5 Points)*

(b) Consider the following **first order logical** formulae

$$\begin{aligned}\varphi_1 &:= \forall x R(x,x)\\ \varphi_2 &:= \forall x \forall y\, R(x,y) \to (\exists z R(x,z) \wedge R(z,y))\\ \varphi_3 &:= \exists x \exists y\, (\neg R(x,y) \wedge \neg R(y,x))\end{aligned}$$

where x, y are variable symbols and R is a binary predicate. Give an interpretation

(i) I_1 which is a **model** of $\varphi_1 \wedge \varphi_2$. *(3 Points)*

(ii) I_2 which is **no model** of $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$. *(2 Points)*

(iii) I_3 which is a **model** of $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$. *(3 Points)*

***Remark**: No proof required.*

Sample Solution

(a) (i) $(\neg x \vee \neg y \vee z \vee w) \wedge (\neg y \vee x) \wedge (\neg z \vee \neg y) \wedge (\neg w \vee \neg y) \wedge y.$

(ii) We use the resolution inference rule to derive an unsatisfiable formula

$$\begin{aligned}\{\neg w, \neg y\}, \{y\} &\vdash_{\mathbf{R}} \{\neg w\}\\ \{\neg z, \neg y\}, \{y\} &\vdash_{\mathbf{R}} \{\neg z\}\\ \{x, \neg y\}, \{y\} &\vdash_{\mathbf{R}} \{x\}\\ \{\neg x, \neg y, z, w\}, \{y\} &\vdash_{\mathbf{R}} \{\neg x, z, w\}\\ \{\neg x, z, w\}, \{\neg w\} &\vdash_{\mathbf{R}} \{\neg x, z\}\\ \{\neg x, z\}, \{\neg z\} &\vdash_{\mathbf{R}} \{\neg x\}\\ \{\neg x\}, \{x\} &\vdash_{\mathbf{R}} \square\end{aligned}$$

(b) (i) Pick $I_1 := \langle \mathbb{R}, \cdot^{I_1} \rangle$ where $R^{I_2}(x,y) :\Longleftrightarrow x \leq_{\mathbb{R}} y$.
This is a model because '$\leq_{\mathbb{R}}$' is *reflexive*, therefore fulfills φ_1. Moreover for every $x, y \in \mathbb{R}$ with $x \leq_{\mathbb{R}} y$ we can choose $z := x$, which fulfills $x \leq_{\mathbb{R}} z \wedge z \leq_{\mathbb{R}} y$. Thus φ_2 is also satisfied.

(ii) Pick $I_2 := \langle \mathbb{R}, \cdot^{I} \rangle$ where $R^{I_2}(x,y) = \texttt{false}$.
This is not a model since it violates φ_1, e.g. $R^{I_2}(5,5) = \texttt{false}$.

(iii) Take two disjoint copies of $\mathbb{R}$ and the standard $\leq_{\mathbb{R}}$ relation on each of them; if x and y are from different copies they are not related in $\mathbb{R}$. Formally let

$$I_3 := \langle \{(a,1) \mid a \in \mathbb{R}\} \dot{\cup} \{(a,2) \mid a \in \mathbb{R}\}, \cdot^{I_3} \rangle$$

where $R^{I_3}((a,g),(b,h)) \Leftrightarrow (g = h$ and $a \leq_{\mathbb{R}} b)$.

This is a model because $\leq_{\mathbb{R}}$ is *reflexive*, therefore I_3 fulfills φ_1. Furthermore for every two $x = (a,g)$ and $y = (b,h)$ with $R^{I_3}((a,g),(b,h))$, i.e., $g = h$, we can choose $z := (a,g)$ which fulfills $R^{I_3}((a,g),(a,g)) \wedge R^{I_3}((a,g),(b,h))$. Thus φ_2 is also satisfied. φ_3 is also satisfied, e.g., $(5,1)$ and $(7,2)$ are incomparable, i.e., we have neither $R^{I_3}((5,1),(7,2))$ nor $R^{I_3}((7,2),(5,1))$